A Fast Algorithm for Optimally Finding Partially Disjoint Shortest Paths

Longkun Guo1, Yunyun Deng1,*, Kewen Liao2,*, Qiang He2, Timos Sellis2 and Zheshan Hu1
1 College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, Fujian, China
2 Department of Computer Science and Software Engineering, Swinburne University, VIC, Australia
lkguo@fzu.edu.cn, {kliao, qhe, tsellis}@swin.edu.au

Abstract
The classical disjoint shortest path problem has recently recalled interests from researchers in the network planning and optimization community. However, the requirement of the shortest paths being completely vertex or edge disjoint might be too restrictive and demands much more resources in a network. Partially disjoint shortest paths, in which a bounded number of shared vertices or edges is allowed, balance between degree of disjointness and occupied network resources. In this paper, we consider the problem of finding $k$ shortest paths which are edge disjoint but partially vertex disjoint. For a pair of distinct vertices in a network graph, the problem aims to optimally find $k$ edge disjoint shortest paths among which at most a bounded number of vertices are shared by at least two paths. In particular, we present novel techniques for exactly solving the problem with a runtime that significantly improves the current best result. The proposed algorithm is also validated by computer experiments on both synthetic and real networks which demonstrate its superior efficiency of up to three orders of magnitude faster than the state of the art.

1 Introduction
Several shortest path problems [Akiba et al., 2015; Hansen and Abdoulahi, 2016; Imamichi et al., 2016] have recently witnessed renewed interests in AI community. Solutions to these problems are largely heuristic-based. In network planning and optimization, the classical disjoint shortest path problem, especially the problem of finding partially disjoint shortest paths, has also started with a heuristic method [Seymour and Kar, 2013] and only recently seen guaranteed optimal solutions [Yallouz et al., 2016]. However, the state-of-art optimal solutions are far from being efficient and scalable, limiting their practicability. In this paper, we investigate a fast solution to the following problem of finding $k$ edge-disjoint shortest paths with a bounded number of common vertices:

\[ \text{Definition 1.} \ (\delta\text{-vertex } k \text{ edge-disjoint shortest path problem, } \delta V-kEDSP): \text{ For a directed graph or network } G = (V, E) \text{ with a source vertex } s \in V \text{ and a destination } t \in V, \text{ a weight function } w : E \rightarrow Z^+, \text{ a given nonnegative integer } \delta \in Z, \delta V-kEDSP \text{ aims to compute } k \text{ edge-disjoint paths } P_1, P_2, \ldots, P_k \text{ in } G, \text{ such that } \sum_{i=1}^{k} \frac{w(P_i)}{\delta} \text{ attains the minimum while among them there are at most } \delta \text{ vertices (besides } s \text{ and } t) \text{ shared by at least two paths.} \]

Intuitively, the above definition constrains edge-disjoint shortest paths intersecting each other with the number of intersection (node) points upper bounded by $\delta$. Compared with the other stricter vertex-disjoint shortest paths problem, $\delta V-kEDSP$ potentially finds a set of paths with a shorter average length. For the problem when $k = 2$, we propose a fast optimal (or namely exact) algorithm to $\delta V$-2EDSP, as we can already envision that the general $\delta V$-kEDSP requires much more algorithmic machinery. Note that throughout the paper we use terms “shortest” and “minimum weight”, and also “exact algorithm” and “optimal algorithm”, interchangeably.

1.1 Related Work
To the best of our knowledge, the $\delta V$-kEDSP problem was first studied in [Yallouz et al., 2016] for $k = 2$, where it was optimally solved within a time complexity $O(mn^2 + n^3 \log n)$. Although the study over $\delta V$-kEDSP is emerging, the case of $\delta = 0$ has already attracted intensive research from mathematicians and computer scientists starting from early 70s. When $\delta = 0$, $\delta V$-kEDSP becomes the complete vertex-disjoint (edge-disjoint) shortest st-path problem, namely the min-sum vertex-disjoint path problem, which can be simply solved with a simple max flow. It was also shown by Suurballe that the problem, for any fixed integer $k > 0$, admits an optimal algorithm with a runtime $O(n^2 \log n)$ [Suurballe, 1974]. Later in [Suurballe and Tarjan, 1984], this runtime was improved to $O\left(\frac{m \log (1 + m/n)}{n}\right)$.

Another important variant of $\delta V$-kEDSP is the Node-Disjoint Path with Congestion ($NDP_wC$) problem. The problem is to maximize the number of paths to respectively connect $k$ given pairs of vertices $s_1t_1, \ldots, s_kt_k$, such that each vertex appears on at most $\gamma$ paths for a given positive integer $\gamma$ that is known as congestion factor [Chekuri et al., 2005; Chekuri and Ene, 2013]. The existing methods mostly modeled and relaxed the $NDP_wC$ problem as...
a natural multicommodity flow problem via linear programming. Chekuri et al. [Chekuri et al., 2005] used the multicommodity flow LP-relaxation to obtain an \( O(\log^2 k \log n) \) approximation with congestion \( \gamma = 4 \) for \( NDPwC \) in planar graph. Then, Chekuri and Ene [Chekuri and Ene, 2013] gave a polynomial time algorithm which routes \( \Omega(OPT) \) pairs with \( O(1) \) congestion in undirected graphs, where \( OPT \) is the value of an optimal fractional solution to a natural multicommodity flow relaxation. For the edge-disjoint case, the edge-disjoint path with congestion \( (EDPwC) \) problem is also to maximize the number of disjoint paths respectively connecting the given pairs of vertices, such that each edge appears on at most \( \gamma \) paths. For the first time addressing \( EDPwC \), Raghavan and Thompson [Raghavan and Thompson, 1987] gave a constant factor approximation with congestion \( \gamma \) relaxed to \( \Omega(\log n/\log \log n) \) by using randomized rounding technique. Later, Andrew presented a randomized \( O((\log^3 n), O((\log^\delta n)^6)) \)-approximation algorithm for \( EDPwC \) in [Andrews, 2010]. Note that for inapproximability, Andrews et al. [Andrews et al., 2005] have also shown that for any congestion \( \gamma = O((\log \log n)/\log \log k) \), there is no \( \log^{\frac{1+\varepsilon}{\gamma}} \) \( n \)-approximation algorithm for \( EDPwC \) unless \( \mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{poly \log \log n}) \), so their poly-logarithmic bi-factor ratio is nearly the best possible. For better congestion guarantee but fewer paths between pairs of vertices, Chuzhoy in [Chuzhoy, 2016] recently gave an efficient randomized algorithm, which routes \( \Omega(OPT) \) demand pairs with congestion of at most 14.

Our techniques for solving \( \delta V-kEDSP \) involve at the last step a transformation to the restricted shortest path (RSP) problem that is known \( \mathsf{NP} \)-complete in [Garey and Johnson, 2002]. To the best of our knowledge, Joksch [Joksch, 1966] was the first to formally solve RSP by presenting a pseudo-polynomial time algorithm using dynamic programming. Although we adopt this pseudo-polynomial algorithm for RSP to form our polynomial time solution (this works because in our case \( \delta \leq n \)), it is worthwhile to mention other related approximations: Hassin gave another pseudo-polynomial algorithm with runtime \( O(m^{2^\varepsilon}) \) for \( OPT \) being the weight of an optimal solution, and then subsequently gave an FPTAS with time \( O\left(m \left(\frac{n^2}{c}\right) \log\left(\frac{\varepsilon}{c}\right)\right)\) [Hassin, 1992]; While Lorenza and Raz [Lorentz and Raz, 2001] proposed another FPTAS with a further improved runtime \( O\left(m n \log \log n + \frac{1}{\varepsilon}\right)\), for any fixed real number \( \varepsilon > 0 \). Recently, the research interest is also on the \( k \)-edge (vertex) disjoint restricted shortest path (KRSP) problem which combines RSP and disjoint paths [Orda and Sprintson, 2004; Guo et al., 2015].

### 1.2 Our Results

In this paper, we propose an exact algorithm for \( \delta V-2EDSP \) with a runtime \( O(\delta m + n \log n) \), which significantly improves the runtime \( O(mn^2 + nn^3 \log n) \) of the previous best solution [Yallouz et al., 2016]. Our algorithm is based on the main observation that, if \( \delta V-2EDSP \) is feasible then there must exist a special optimal solution, what we called a spiral optimal solution. The algorithm progresses with the idea of path augmentation. It first computes a shortest \( st \)-path, then this path gets augmented to a pair of spiral \( st \)-paths via our specially constructed residual graph with an extra cost function to capture the number of shared vertices. Moreover, we show that if \( \delta V-2EDSP \) is feasible, then there must exist a spiral optimal solution whose shared vertices are all on the shortest \( st \)-path. With techniques from graph theory and network flow theory, we prove that our algorithm produces an optimal solution to \( \delta V-2EDSP \). The proposed algorithm has the potential to extend to \( \delta V-kEDSP \) for general \( k \), but we omit this discussion here due to space limit. We also run experiments on both synthetic and real networks to validate the practical performance of our algorithm against other baselines. Experimental results show an up to three orders of magnitude runtime reduction – matching our theoretical runtime analysis.

### 2 Exact Algorithms for \( \delta V-2EDSP \)

In this section, we first focus on an important property of \( \delta V-2EDSP \): there must exist a special optimal solution called spiral optimal solution if an instance of \( \delta V-2EDSP \) is feasible. Then we give an algorithm to compute such a spiral optimal solution, which equivalently solves \( \delta V-2EDSP \).

#### 2.1 Spiral Optimal Solutions to \( \delta V-2EDSP \)

We denote a network by a digraph \( G = (V, E) \), where \( V \) and \( E \) are respectively the sets of vertices and edges. Assume that \( n = |V| \) and \( m = |E| \). Given an \( st \)-path \( P \) and two vertices \( u, v \in P \), we denote by \( P(u, v) \) the subpath of \( P \) from \( u \) to \( v \), and say \( u \prec v \) if and only if \( v \neq u \) appears on \( P(u, v) \), i.e., \( v \in P(u, v) \). For two paths \( P_1, P_2 \), we say \( P_1(u, v) \) is a maximal segment within \( P_1 \cap P_2 \) if (1) \( P_1(u, v) \subseteq P_1 \cap P_2 \) and (2) there exists in \( P_1 \cap P_2 \) no other segment that contains \( P_1(u, v) \), that is, there exists no \( P_1(x, y) \subseteq P_1 \cap P_2, i \in \{1, 2\} \), such that \( P_1(x, y) \supseteq P_1(u, v) \) holds.

Our algorithm will actually compute a special optimal solution called a spiral optimal solution, which is formally as follows (An example comparing a spiral optimal solution vs a non-spiral optimal solution is depicted in Figure 1):

**Definition 2.** Let \( P^* \) be a shortest \( st \)-path in \( G \). Then an optimal solution to \( \delta V-2EDSP \), say \( P = \{P_1^*, P_2^*\} \), is a spiral optimal solution if the following two conditions both hold: (1) For any \( u, v \in P^* \cap P_i^*, i \in \{1, 2\}, u \prec \gamma, v \) if and only if \( v \neq u \) appears on \( P(u, v) \), i.e., \( v \in P(u, v) \). For two paths \( P_1, P_2 \), we say \( P_1(u, v) \) is a maximal segment within \( P_1 \cap P_2 \) if (1) \( P_1(u, v) \subseteq P_1 \cap P_2 \) and (2) there exists in \( P_1 \cap P_2 \) no other segment that contains \( P_1(u, v) \), that is, there exists no \( P_1(x, y) \subseteq P_1 \cap P_2, i \in \{1, 2\} \), such that \( P_1(x, y) \supseteq P_1(u, v) \) holds.

Note that \( P_1^* \) and \( P_2^* \) in the above definition must be edge-disjoint and hence can only share common vertices. We say \( v_j, v_j \) are properly ordered if \( v_j \prec v_j \) satisfy Condition 1; Otherwise, say \( v_j \prec v_j \) on \( P^* \) while \( v_j \prec v_j \) on \( P_i^*, i \in \{1, 2\} \), and \( v_j \prec v_j \) is antiderior. Note that there may exist \( P_i^* \in P, i \in \{1, 2\} \), such that \( P^* \supseteq P_i^* \).

The key idea of our algorithm is inspired by the Simplex method of solving linear programs, in which a special optimal solution called basic optimal solution, rather than any optimal solution is computed. So similar to the correctness proof of the Simplex method, we need first to show it suffices.
to compute a spiral optimal solution to solve $\delta$V-2EDSP, as given in the following theorem:

**Theorem 3.** If an instance of $\delta$V-2EDSP is feasible, then it must have spiral optimal solutions.

**Proof.** For Condition 1 of Definition 2, we show that if an instance of $\delta$V-2EDSP is feasible, then an optimal solution can be constructed satisfying the condition. Assume $\{P_1^*, P_2^*\}$ is an optimal solution of $\delta$V-2EDSP, for which there exist $v_j, v_j' \in P^* \cap P_2^*$, $i \in \{1, 2\}$, such that $v_j \prec v_j'$ on $P^*$ but $v_j' \prec v_j$ on $P_2^*$. Then we can construct a pair of disjoint path $P_1$ and $P_2$ of which the number of antidertered vertex pairs decreases at least one comparing to $\{P_1^*, P_2^*\}$. Consequently, an optimal solution without any antidertered vertex pair can be constructed by repeatedly decreasing the number of antidertered vertex pairs.

It remains to give the construction of $P_1$ and $P_2$. Without loss of generality, we assume $v_j$ is the vertex closest to $t$ in all antidertered vertex pairs. Let $u_i$, which belongs to $P_1 \cup P_{3-i}$, be the first vertex after $v_j$. We will construct $P_1$ and $P_2$ for both two cases that $u_1 \in P_1^*$ or $u_1 \in P_{2}^*$.

1. If $u_1 \in P_1^*$, let $w \in P_{2}^*$ be the last vertex before $v_j$. We will show that there exists a solution $P_1$ and $P_2$ whose weight is not larger than that of $P_1^*$ and $P_2^*$, but with fewer antidertered vertex pairs. We need only to simply set $P_1 = P_1^*(s, v_j) \cup P^*(v_j, u) \cup P_{3-i}^*(u, t)$ and $P_2 = P_2^*(s, w) \cup P^*(w, v_j) \cup P_i^*(v_j, t)$. Apparently, every properly ordered vertex pair in $\{P_1^*, P_2^*\}$ remains so while the antidertered vertex pair $\{v_j, v_j\}$ becomes also properly ordered in $P_1$ and $P_2$. That is, the number of antidertered vertex pairs decreases at least one from $\{P_1^*, P_2^*\}$ to $\{P_1, P_2\}$. It remains only to show the weight of $\{P_1, P_2\}$ is not larger than $\{P_1^*, P_2^*\}$, which is to show $w(P^*(v_j, u) \cup P_i^*(u, t)) \leq w(P_1^*(s, v_j) \cup P_1^*(v_j, u) \cup P_{3-i}^*(u, t))$. Suppose otherwise, i.e., $w(P^*(v_j, u) \cup P_i^*(u, t)) > w(P_1^*(s, v_j) \cup P_1^*(v_j, u) \cup P_{3-i}^*(u, t))$. Then $P_1 = P_1^*(s, v_j) \cup P_i^*(u, t)$ is an $s$-path with a smaller weight than $P_1^*$, contradicting with the optimality of $P_1^*$. Therefore, the two constructed paths $P_1$ and $P_2$ are with a weight that is not larger than $\{P_1^*, P_2^*\}$.

2. If $u_1 \in P_2^*$, we will construct a new path $P_1$, which is with a weight no larger than $P_2^*$ but at least one less antidertered vertex pair. For the task, we need only to simply set $P_1 = P_1^*(s, v_j) \cup P^*(v_j, u) \cup P_2^*(u, t)$. Apparently, $w(P_1) \leq w(P_1^*)$, because $w(P^*(v_j, u)) \leq w(P_1^*(v_j, u))$. Besides, since $v_j \notin P_1$, $P_1$ is with at least one less antidertered vertex pair than $P_1^*$. Moreover, according to the construction of $P_2$, the path pair $\{P_1^*, P_{3-i}^*\}$ shares only common vertices of the original $\{P_1^*, P_{3-i}^*\}$, and hence $\{P_1, P_{3-i}\}$ shares at most the same number of common vertices as $\{P_1^*, P_{3-i}^*\}$ which is bounded by $\delta$.

For Condition 2, let $\{P_1^*, P_2^*\}$ be an optimal solution to $\delta$V-2EDSP that already satisfies Condition 1. Then we show an optimal solution also satisfying Condition 2 can be constructed from $\{P_1^*, P_2^*\}$. Let $v_j$ and $v_j'$ be two segments in which all the interior vertices belong to $S_1 = \{seg | seg \subseteq P_1^* \cap P_2^*\}$. Let $u$ be the first vertex in $seg_j$. Assume that there exists no segment of $S_2 = \{seg | seg \subseteq P_1^* \cap P_2^*\}$ that appears on $P^*$ between $seg_j$ and $seg_j'$. Then let $P_1 = P_1^*(s, u) \cup P^*(u, v) \cup P_2^*(v, t)$, which is $P_1^*$ excepting replacing $P_1^*(u, v)$, the subpaths of $P_1^*$ between $seg_j$ and $seg_j'$, by $P^*(u, v)$. Apparently, $P^*(u, v)$ is with a weight no larger than $P_1^*(u, v)$, so $w(P_1) \leq w(P_1^*)$. Moreover, $seg_j$ and $seg_j'$ are merged into one segment on $P_1$. Besides, $P_1$ and $P_2^*$ share only common vertices of $P_1^*$ and $P_2^*$. Therefore, by repeating such operations, clearly such a $\{P_1^*, P_2^*\}$ can eventually become a solution to $\delta$V-2EDSP satisfying both Condition 1 and Condition 2.

Note that, Condition 2 will not hold trivially without Condition 1. Because in that case we could not simply use $P_n^*(u, v)$ to connect $seg_j$ and $seg_j'$, as $seg_j \prec_{P_1^*} seg_j'$ may hold. That is why we prove Condition 1 before Condition 2. Moreover, because $P_1^*$ and $P_2^*$, as a spiral optimal solution to $\delta$V-2EDSP, are allowed to share $\delta$ common vertices, we have further observation on the relationship between their common vertices and $P_1^*$ as below:

**Lemma 4.** If an instance of $\delta$V-2EDSP is feasible, there must exist a spiral optimal solution $P_1^*$ and $P_2^*$ wrt $P^*$ such that the common vertices of $P_1^*$ and $P_2^*$ are all on $P^*$, i.e. $(V(P_1^*) \cap V(P_2^*)) \subseteq P^*$.

**Proof.** Suppose otherwise, there exists a common vertex of $P_1^*$ and $P_2^*$ that does not belong to $P^*$, say $p \in P_1^* \cap P_2^*$ but $p \notin P^*$. We will show that there exists a solution $P_1$ and $P_2$ whose weight is not larger than $P_1^*$ and $P_2^*$ but shares one less common vertex. Let $w, z \in P_1^* \cap P_2^*$ and $u, v \in P^* \cap P_2^*$ be the two pairs of vertices nearest to $p$ on $P_1^*$ and $P_2^*$, respectively. W.l.o.g., we assume $w < z$ on $P_1^*$ and $u < v$ on $P_2^*$; then $P_1$ and $P_2$ are as follows: $P_1 = s \rightarrow w \rightarrow \cdots \rightarrow z \rightarrow \cdots \rightarrow t$ and $P_2 = u \rightarrow \cdots \rightarrow v \rightarrow \cdots \rightarrow t$.

Following Condition 1 of Definition 2, there are six different permutations of $u, v, w$ and $z$ appearing on $P^*$: (1) $w < u < v < z$; (2) $w < u < z < v$; (3) $w < z < u < v$; (4) $u < w < u < v$; (5) $u < w < z < v$; (6) $u < v < w < z$.

We need only to consider Case (1)-(3), since Case (4)-(6) are symmetrically similar. In the following, we will show in Case (1), $P_1^*$ and $P_2^*$ can not be a spiral optimal solution; while in Case (2) and (3), the number of common vertices decreases at least one:

1. $w < u < v < z$ on $P^*$: By Condition 2 of the definition of a spiral optimal solution, there must be at least a vertex $x \in P_1^*$ appearing on $P^*$ between $u$ and $v$, i.e. $x \in P^*(u, v) \setminus \{u, v\}$. Then because $w$ and $z$ are two vertices of $P^* \cap P_1^*$ that
Algorithm 1 Construction of a split-residual graph.

**Input:** A digraph \( G(V, E) \), specified vertices \( s \) and \( t \), a weight function \( w(e) \) and a shortest \( st \)-path \( P^* \).

**Output:** A split-residual graph \( \hat{G} \).

1. Set \( \hat{G} := G \setminus \{(V(P^*) \setminus \{s, t\}) \) where each edge \( e \in \hat{G} \) is with \( c(e) := 0 \) and a weight equal to its weight in \( G \);
2. **For** each interior vertex \( v \in P^* \setminus \{s, t\} \) **do**
   3. Add two vertices \( v_1, v_2 \) to \( G \);
   4. Add two edges to \( \hat{G} \) edge \( e(v_1, v_2) \) with weight 0 and cost 1 and edge \( e(v_2, v_1) \) with weight 0 and cost 0;
5. **Endfor**
6. **For** each edge \( e(u, v) \) in \( G \) **do**
7. **If** \( e(u, v) \) is on \( P^* \) **then**
   8. Set \( \hat{G} := \hat{G} \cup \{e(v_1, u_2)\}, c(e(v_1, u_2)) := 0 \)
   9. Else
10. **If** \( u \in P^* \setminus \{s, t\} \) **then**
11. Set \( \hat{G} := \hat{G} \cup \{e(u_2, v)\}, c(e(u_2, v)) := 0 \)
12. **EndIf**
13. **Return** \( G \).

are closest to \( p \), \( x \) can only belong to \( P_1^*(s, w) \) or \( P_1^*(z, t) \). However, there will be an antiderived vertex pair \( (w, x) \) for \( x \in P_1^*(s, w) \) and \( (z, x) \) for \( x \in P_1^*(z, t) \). This contradicts with the fact that \( (P_1^*, P_2^*) \) is a spiral optimal solution.

The difficulty of the algorithm is mainly in the construction of a split-residual graph \( \hat{G} \) with respect to \( G \) and \( P^* \), such that \( P^* \oplus Q \) is exactly an optimal solution to \( 0 \mathrm{V2EDSP} \), where \( Q \) is a shortest \( st \)-path with cost at most \( \delta \) in \( G \), capturing the corresponding edge-disjoint path pair in the original graph \( G \) that is with at most \( \delta \) common vertices. Here we use \( P \oplus Q \) to denote the set of edges \( P \cup Q \) but with all opposite parallel edge pairs removed, where an opposite parallel edge pair is a pair of edges between an identical vertex pair but in opposite direction. The construction of the split-residual graph is similar to constructing a residual graph, except that every interior vertex on \( P^* \) is split and the edges are modified accordingly.

**Definition 5. (Residual graph)** Given a graph \( G \) with a weight function \( w : E \rightarrow \mathbb{Z} \), \( P^* \) is a shortest \( st \)-path in \( G \), the residual graph \( \hat{G} \) is constructed as follows. For each edge \( (u, v) \) \( \in G \setminus P^* \), we add to \( \hat{G} \) an edge of the same weight as in \( G \). For each edge \( (u, v) \) \( \in P^* \), we add to \( \hat{G} \) a reversed edge \( (u, v) \) with a weight \( -w((u, v)) \).

The main steps are as below. Firstly, construct a traditional residual graph \( G := G \setminus \mathcal{P} \setminus P^* \), where \( \mathcal{P} \) is the set of edges resulted from every edge on \( P^* \) with direction reversed and weight set negative, i.e. for every edge \( e = (u, v) \) \( \in P^* \), add \( e' = (v, u) \) to \( \mathcal{P} \) with a weight \( w(e') = -w(e) \). Secondly, split each interior vertex of \( P^* \), say \( v \), into two vertices \( v_1 \) and \( v_2 \), with an edge \( (v_1, v_2) \) with weight 0 and cost 1 and an edge \( (v_2, v_1) \) with weight 0 and cost 0, where \( Q \) going through edge \( (v_1, v_2) \) indicates that in \( G \) the corresponding paths resulting from \( P^* \oplus Q \) will share the vertex \( v \). Then redefine the edges that containing \( v \): (1) replace each edge \( (u, v) \) \( \in P^* \) with \( (u, v_1) \) \( \in \mathcal{P} \); (2) replace each edge \( (x, u) \) \( \notin \mathcal{P} \) with \( (x, v_2) \) \( \in \mathcal{P} \); (3) replace each edge \( (v, y) \) \( \notin \mathcal{P} \) with \( (v_2, y) \) \( \in \mathcal{P} \). Moreover, the redirected edges are with cost 0 and the same weight as the original edges. Note that the above splitting can also deal with edge \( (x, y) \) with either \( x \in \mathcal{P} \) or \( y \in \mathcal{P} \). For brevity, we denote by \( E_s \) the set of cost 1 edges.

The detailed construction is formally as in Algorithm 1 (An example of splitting vertices is depicted as in Figure 2).

**Proposition 6.** Let \( \{P_1^*, P_2^*\} \) be a spiral optimal solution to \( 0 \mathrm{V2EDSP} \) wrt \( G \) and \( P^* \). Let \( H = P_1^* \cup P_2^* \cup P^* \), \( \hat{H} \) be the split residual graph for \( H \) wrt \( P^* \). Assume that \( E_s \subseteq \hat{H} \).
is the set of edges between the pairs of split vertices for each \( v \in P^* \). Then there must exist in \( H \) an \( st \)-path \( Q \), for which \( c(Q) \leq \delta \) and \( w(P^*) + w(Q) \leq w(P^*_1) + w(P^*_2) \) both hold.

**Proof.** Following Lemma 4, the set of common interior vertices of \( P^*_1 \) and \( P^*_2 \), say \( V(P^*_1 \cap P^*_2) \setminus \{ s, t \} \) = \( \{ v^1, v^2, \ldots, v^h \} \), must be all on the shortest path \( P^* \). Then \( P^*_1(v^h, v^{h+1}) \) and \( P^*_2(v^h, v^{h+1}) \), \( h = 0, \ldots, \delta \), is a pair of vertex disjoint paths between \( v^h \) and \( v^{h+1} \) in \( H \). We will show that there exists a path \( Q^h \subseteq H \) between \( v^h_2 \) and \( v^{h+1}_2 \) in \( H \) with only edges of cost 0. Then there exists an \( st \)-path \( Q = sQ^0_1v^1_2Q^1_2v^2_2 \ldots v^h_2Q^h_2v^{h+1}_2 \) whose total cost is \( \delta \) as edge \( (v^h_1, v^h_2) \), \( h = 1, \ldots, \delta \), is with cost 1 while every other edge on the path is with cost 0.

It remains only to construct \( Q^h \), \( \forall h \). We denote the set of maximal segments in \( P^* \cap P^*_1 \) \( (i = 1, 2) \) by \( S_i = \{ P^*_{(x_i, j, y_i, j)} \ | \ j = 1, \ldots, h_i \} \). W.l.o.g. assume that the first and the last maximal segments both belong to \( P^*_2 \), i.e. \( x_{2,j} = v^h_1 \) and \( y_{2,h_1} = v^{h+1}_2 \). Note that \( (x_{2,j}, y_{2,j}) \) is exactly the previous maximal segment before \( (x_1, y_1) \).

Then we can construct \( Q^h \) by using edges in \( H \) only:

\[
Q^h = \bigcup_{j=1}^{h_1} \left[ P^*(x_{1,j}, y_{2,j}) \cup P^*_2(y_{2,j}, x_{2,j+1}) \right.
\]

\[
\left. \cup P^*(x_{2,j+1}, y_{1,j}) \cup P^*_1(y_{1,j}, x_{1,j+1}) \right] \cup P^*(v^h_1, v^h_2, v^h_1)
\]

where \( x_{1,h_2+1} = t \). Because the constructed \( Q^h \) contains only segments of \( P^*_1 \cup P^*_2 \) and \( P^* \) whose edges are of cost 0, \( Q \) is an \( st \)-path in \( H \) of cost at most \( \delta \). Besides,

\[
\bigcup_{j=1}^{h_1} \left[ P^*(x_{1,j}, y_{2,j}) \cup P^*(x_{2,j+1}, y_{1,j}) \right]
\]

are exactly the set of edges resulting from reversing edges of \( P^* \setminus (S_1 \cup S_2) \). Therefore, we have \( w(P^*) + w(Q) = w(S_1) + w(S_2) + w(P^* \setminus S_1) + w(P^*_2 \setminus S_2) = w(P^*_1) + w(P^*_2) \). This completes the proof.

**Lemma 7.** Let \( Q \) be a shortest \( st \)-path with cost bounded by \( \delta \) in \( G \) wrt \( G^* \) and \( P^* \), where \( P^* \) is a shortest \( st \)-path in \( G \). Assume that \( Q^* \) is a Q-except contracting every edge between each pair of split vertices. Then \( P^* \ominus Q^* \) is exactly the union of two edge disjoint paths between \( s \) and \( t \), which have a minimum total weight and share at most \( \delta \) common vertices.

**Proof.** We will firstly show that \( P^* \ominus Q^* \) is exactly composed by two edge disjoint \( st \)-paths of \( G \), which shares at most \( \delta \) common vertices; secondly that \( w(P^* \ominus Q^*) \leq w(\text{OPT}) \), where \( \text{OPT} \) is an optimal solution to 4\( V \)-EDSP.

For the first, let \( Q^* \) be \( Q \) except contracting every edge between every pair of split vertices. We need only to show the edges of \( P^* \ominus Q^* \) exactly compose two edge disjoint \( st \)-paths in \( G \). First, \( P^* \cup Q^* \) contains an \( st \)-flow of value 2, since \( s \) and \( t \) are respectively with a degree 2 and \(-2 \) while every other vertex of \( P^* \cup Q^* \setminus \{ s, t \} \) is with degree 0. Then because \( P^* \cup Q^* \) is \( P^* \cup Q^* \) but removing all parallel edge pairs, \( P^* \ominus Q^* \) contains only edges in \( G \). Moreover, each removed parallel edge pair is actually a cycle where each vertex is of degree 0, so in \( P^* \ominus Q^* \), \( s \) and \( t \) respectively remain degree 2 and \(-2 \), while every other vertex remains degree 0. Therefore, \( P^* \ominus Q^* \) is an \( st \)-flow of value 2, and hence is a pair of edge disjoint \( st \)-path in \( G \) because each edge in \( P^* \ominus Q^* \) is integral.

Then we can construct \( Q^* \) by \( \delta V \)-EDSP, say \( \{ P^*_1, P^*_2 \} \). Let \( H = P^*_1 \cup P^*_2 \cup P^* \) and \( \hat{H} \) be the accordingly residual graph wrt \( P^* \).

Then from Proposition 6, we can obtain an \( st \)-path \( Q^* \) in \( \hat{H} \) with \( c(Q^*) \leq \delta \) and \( w(P^*) + w(Q^*) \leq w(P^*_1) + w(P^*_2) \).

Hence, \( H \subseteq \hat{G} \), \( Q^* \) is also an \( st \)-path in \( \hat{G} \) with both \( c(Q^*) \leq \delta \) and \( w(P^*) + w(Q^*) \leq w(P^*_1) + w(P^*_2) = w(\text{OPT}) \). Then because \( Q \) is the shortest path in \( G \), we have \( w(Q) \leq w(Q^*) \), and hence \( w(P^*) + w(Q) \leq w(\text{OPT}) \) holds. This completes the proof.

Following the above construction, we can obtain a split-residual graph \( \hat{G} = (V, \hat{E}) \), where the \( \delta V \)-EDSP problem is transformed to the problem of finding a minimum weight \( st \)-path with cost bounded by \( \delta \), namely the Binary Restricted Shortest Path (BRSP) problem, which is clearly a special case of the restricted shortest path (RSP) problem. Following the algorithm for RSP [Joksch, 1966], the BRSP problem can be solved in time \( O(\delta |E|) \) by employing the dynamic programming method. Thus, \( Q \) can be computed within time \( O(\delta |E|) \). Then following Lemma 7, \( P^* \ominus Q^* \) exactly composes an optimal solution to \( \delta V \)-EDSP, where \( Q^* \) is the shortest edge pair of split vertices back to one vertex in \( G \), say contracting \( v_1 \) and \( v_2 \) to \( n \). The formal layout of the whole algorithm is as in Algorithm 2.

**Algorithm 2** An exact algorithm for \( \delta V \)-EDSP.

**Input:** A digraph \( G(V, E) \), source \( s \) and destination \( t \), a weight function \( w(e) \), and \( \delta \in Z^+ \).

**Output:** A solution \( \{ P_1, P_2 \} \) to \( \delta V \)-EDSP.

1: Compute a shortest path \( P^* \) by Dijkstra’s algorithm [Cormen, 2009];
2: Construct a split-residual graph \( \hat{G} \) wrt \( G \) and \( P^* \) by Algorithm 1;
3: Find a shortest path \( Q \) in \( \hat{G} \) with \( c(Q) \leq \delta \) by employing the algorithm for RSP as in [Joksch, 1966];
4: Decompose \( P^* \ominus Q \) into two paths \( P_1 \) and \( P_2 \);
5: Return \( P_1 \) and \( P_2 \).
the bound of the number of common vertices. Other steps obviously takes trivial time comparing to the above ones. Therefore, the total runtime of Algorithm 2 is $O(\delta m + n \log n)$. 

Combining Lemma 7 and 8, we immediately have the following theorem:

**Theorem 9.** Algorithm 2 runs in time $O(\delta m + n \log n)$ and outputs an optimal solution to $\delta V$-2EDSP.

## 3 Experimental Results

In this section, we shall first evaluate the practical performance of Algorithm 2 (the exact algorithm, denoted as EA) by comparing its runtime and solution quality with another two algorithm baselines: the best previous algorithm (PA) presented in [Yallouz et al., 2016], and an exact algorithm via solving a flow integer linear program (ILP) formulation$^1$ we designed for $\delta V$-2EDSP. Then we show EA is scalable by running it against real-world network data including communication networks and P2P networks from Stanford Large Network Dataset Collection$^2$ within Stanford Network Analysis Project (SNAP). These algorithms are implemented in Python 2.7 (The code is available upon request), on a PC with Intel Core i5 processor, and 8GB memory. Our implementation adopts the NetworkX library$^3$ to construct and process graphs, the GLPK library$^4$ to solve the ILP.

### 3.1 Simulation Results

In this subsection, we evaluate the runtime of the algorithms by simulation experiments, in which we generate a random graph from NetworkX with $n$ vertices, and $m$ edges each with a weight uniformly distributed in $[1, 100]$. The number of allowed common vertices is the parameter $\delta$.

Simulation results comparing runtimes of EA, PA and ILP in seconds are depicted in Table 1. For each row of the table, we first randomly generated 1000 different graphs of the given size $(n, m)$, then respectively ran the three algorithms against each graph given a randomly chosen source and destination node pair, and at last computed the average time over the 1000 runs. Notice that during runs, the encountered infeasible graph instances do not contribute to our average time calculations. In the experiments, when the problem size is not larger than $n = 100$ and $m = 1000$, the runtime of EA is close to that of ILP$^5$, but significantly faster than PA, say, about a thousand times faster. This matches the theoretical runtime difference between EA and PA, i.e. $O(\delta m + n \log n)$ vs $O(mn^2 + n^3 \log n)$. When the problem size grows larger, PA quickly becomes not scalable as it takes too long to produce a solution. So for larger graphs we only compared the runtimes of EA and ILP. The simulation results show that EA

$^1$The formulation is available upon request but was omitted here due to space limit.
$^2$http://snap.stanford.edu/data/
$^3$https://networkx.github.io/
$^4$https://www.gnu.org/software/glpk/
$^5$Most likely because the random graph structure with a small $n$ becomes easier for the solution search of ILP solver that usually runs in worst case exponential time in $n$.

<table>
<thead>
<tr>
<th>Size $(n, m, \delta)$</th>
<th>EA (s)</th>
<th>PA (s)</th>
<th>ILP (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(50, 250, 5)</td>
<td>0.023</td>
<td>7.843</td>
<td>0.029</td>
</tr>
<tr>
<td>(55, 302, 6)</td>
<td>0.026</td>
<td>10.404</td>
<td>0.031</td>
</tr>
<tr>
<td>(60, 3580, 6)</td>
<td>0.03</td>
<td>14.515</td>
<td>0.032</td>
</tr>
<tr>
<td>(65, 422, 6)</td>
<td>0.033</td>
<td>20.381</td>
<td>0.034</td>
</tr>
<tr>
<td>(70, 499, 6)</td>
<td>0.035</td>
<td>26.468</td>
<td>0.035</td>
</tr>
<tr>
<td>(75, 562, 8)</td>
<td>0.041</td>
<td>34.255</td>
<td>0.047</td>
</tr>
<tr>
<td>(80, 640, 8)</td>
<td>0.044</td>
<td>43.820</td>
<td>0.051</td>
</tr>
<tr>
<td>(85, 722, 8)</td>
<td>0.053</td>
<td>54.767</td>
<td>0.061</td>
</tr>
<tr>
<td>(90, 810, 10)</td>
<td>0.075</td>
<td>68.818</td>
<td>0.074</td>
</tr>
<tr>
<td>(95, 902, 10)</td>
<td>0.080</td>
<td>85.434</td>
<td>0.084</td>
</tr>
<tr>
<td>(100, 1000, 10)</td>
<td>0.099</td>
<td>104.518</td>
<td>0.099</td>
</tr>
</tbody>
</table>

Table 1: Runtime analysis on randomly generated graphs.

still runs much faster than ILP and more importantly the gap between these two grows with graph size.

It is also worth mentioning that, when testing against the same graph, the min-sum weights of the solutions, produced by all three algorithms, coincide in all experiments on the generated thousands of graphs. It was known that ILP and PA always output optimal solutions, so this observation can be regarded as an experimental evidence that our EA also produces optimal solutions, complementing our optimality proof.

### 3.2 Real-world Evaluation

Besides simulation results, we also ran EA against some real-world network datasets from the SNAP collections. In the experiment, we ran EA for 200 times and each time we instead randomly pick two vertices in the network as source and destination. We again report the average runtime for EA. We also reasonably set $\delta = 10$, because in real networks, many protocols (TCP/IP etc.) only allow a data package pass through at most 16 routers from source to destination [Kurose and Ross, 2009]. Therefore a shortest path for transmitting data will have at most 16 interior vertices in most cases.

The experimental results depicted in Table 2 show that, roughly the runtime of EA is linearly dependent on the number of edges of the network. Moreover, EA takes up to 35 seconds to process a network with 265k nodes and 420k edges whereas PA and ILP in our experiments can not even complete in many hours. Also noting that, although currently EA takes tens of seconds to process median size networks, it can be significantly accelerated with a more sophisticated C/C++ implementation and a higher end computing platform. These together will allow EA to efficiently process much larger graphs with hundreds of millions of nodes and edges. This is also supported from the time complexity of EA that is almost linear in the network size.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Nodes</th>
<th>Edges</th>
<th>EA (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email-Konect</td>
<td>36692</td>
<td>103831</td>
<td>20.76</td>
</tr>
<tr>
<td>Email-Konect</td>
<td>36692</td>
<td>103831</td>
<td>20.76</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>96284</td>
<td>33839</td>
<td>2.53</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>96284</td>
<td>33839</td>
<td>2.53</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>8717</td>
<td>33252</td>
<td>2.94</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>8717</td>
<td>33252</td>
<td>2.94</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>6301</td>
<td>20777</td>
<td>1.92</td>
</tr>
<tr>
<td>p2p-Gnut.09</td>
<td>6301</td>
<td>20777</td>
<td>1.92</td>
</tr>
</tbody>
</table>

Table 2: Runtime analysis against network datasets from SNAP.
4 Conclusion
In this paper, we carried on studying the problem of finding minimum-weight \( k \) edge-disjoint partially vertex-disjoint paths, namely \( \delta V \)-E\( k \)EDSP. We proposed an optimal algorithm for \( \delta V \)-E\( k \)EDSP that efficiently runs in \( O(mn + n \log n) \) time and in experiments demonstrates a significant runtime gain. This greatly improves the previous best runtime bound of \( O(mn^2 + n^2 \log n) \) [Yallouz et al., 2016], where \( m \) and \( n \) are respectively the number of edges, vertices and allowed shared vertices. Note that the time complexity of our algorithm flexibly depends on \( \delta \), which is at most \( n \) (though a fixed \( \delta \) is more practical), so the worst-case complexity is \( O(mn + n \log n) \) still a much better bound. We are currently investigating solving \( \delta V \)-kE\( k \)EDSP by extending our technique for \( \delta V \)-E\( k \)EDSP. Also we are considering the counterpart problem \( \delta E \)-kE\( k \)EDSP, in which the number of edges shared by at least two paths is bounded by \( \delta \) instead.

Acknowledgements
The research is supported by Natural Science Foundation of China (Nos. 61772005, 61300025) and Natural Science Foundation of Fujian Province (No. 2017J01753).

References


